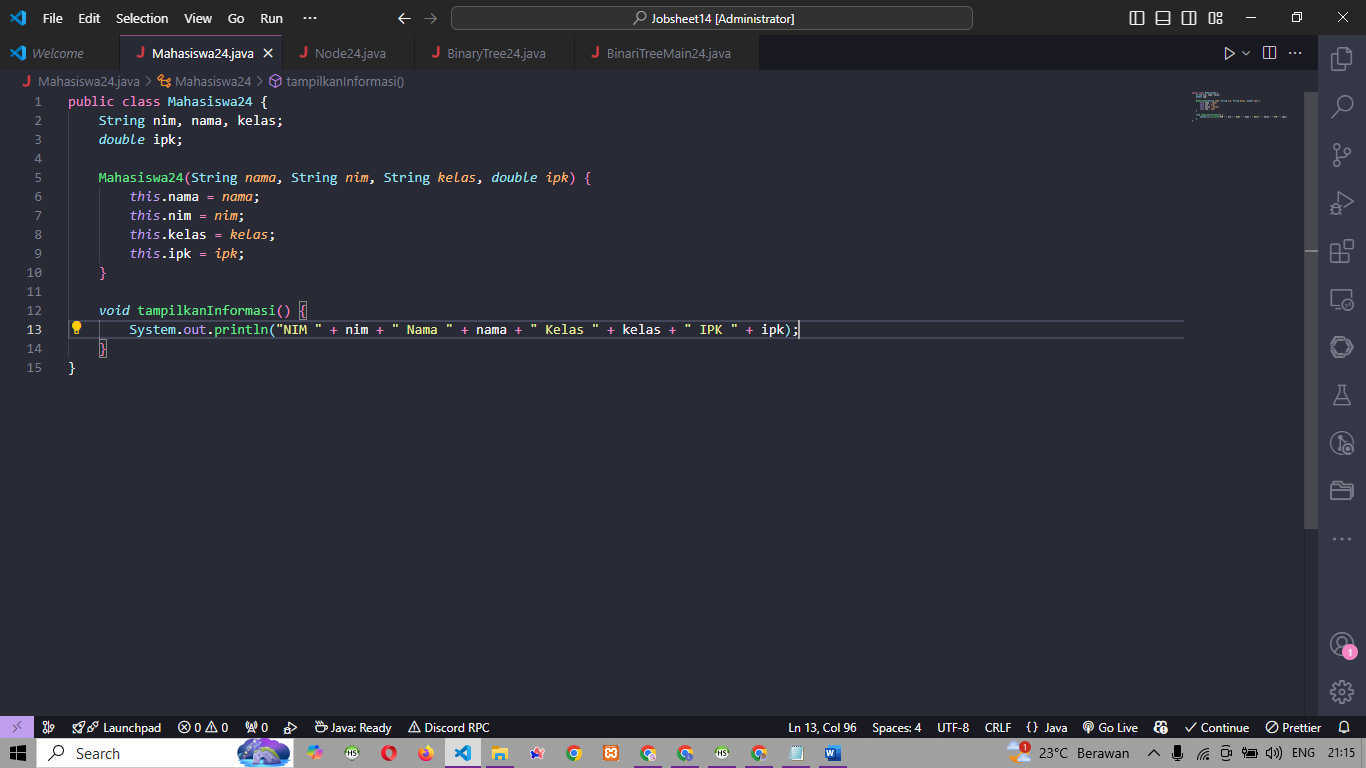
**Nama: Wahda Adella Putri Febriana**

**Kelas: 1B / 244107020156**

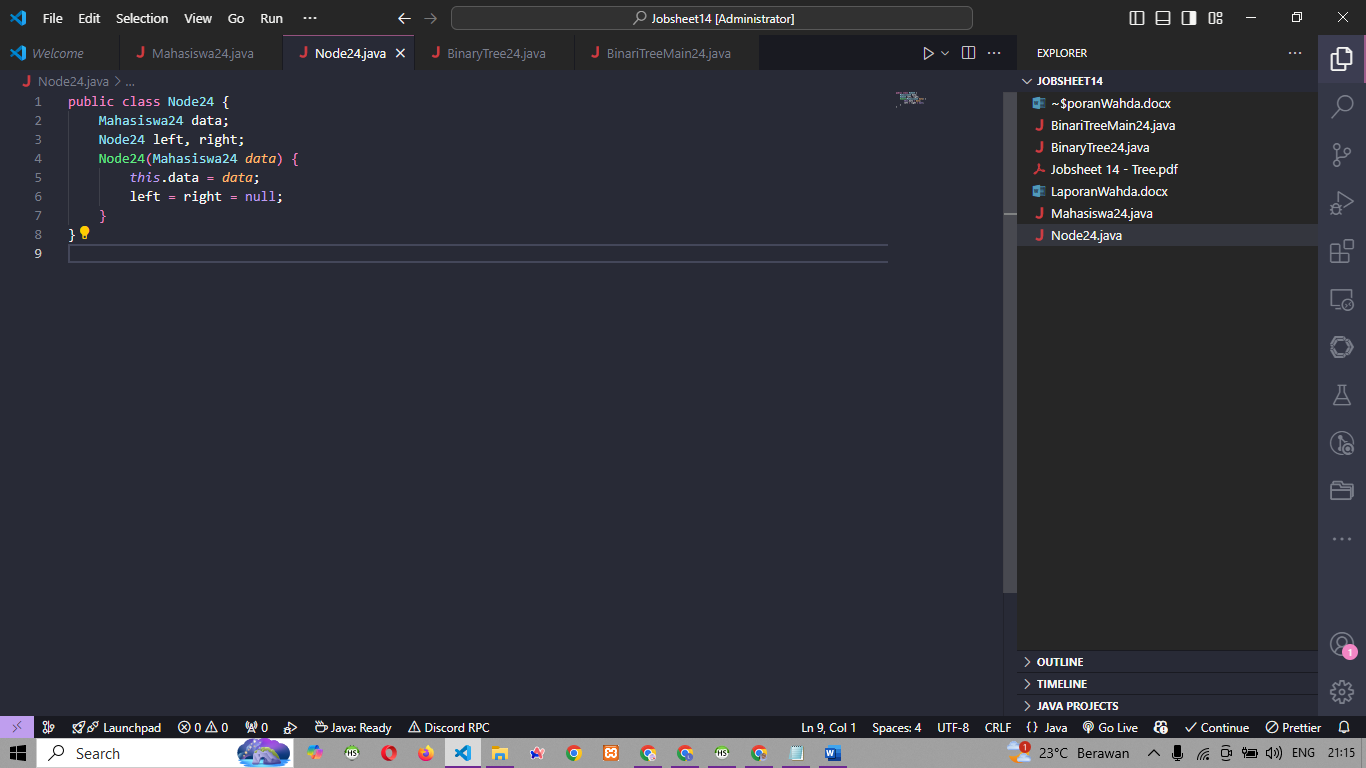
**Absen: 24**

**Percobaan 1**

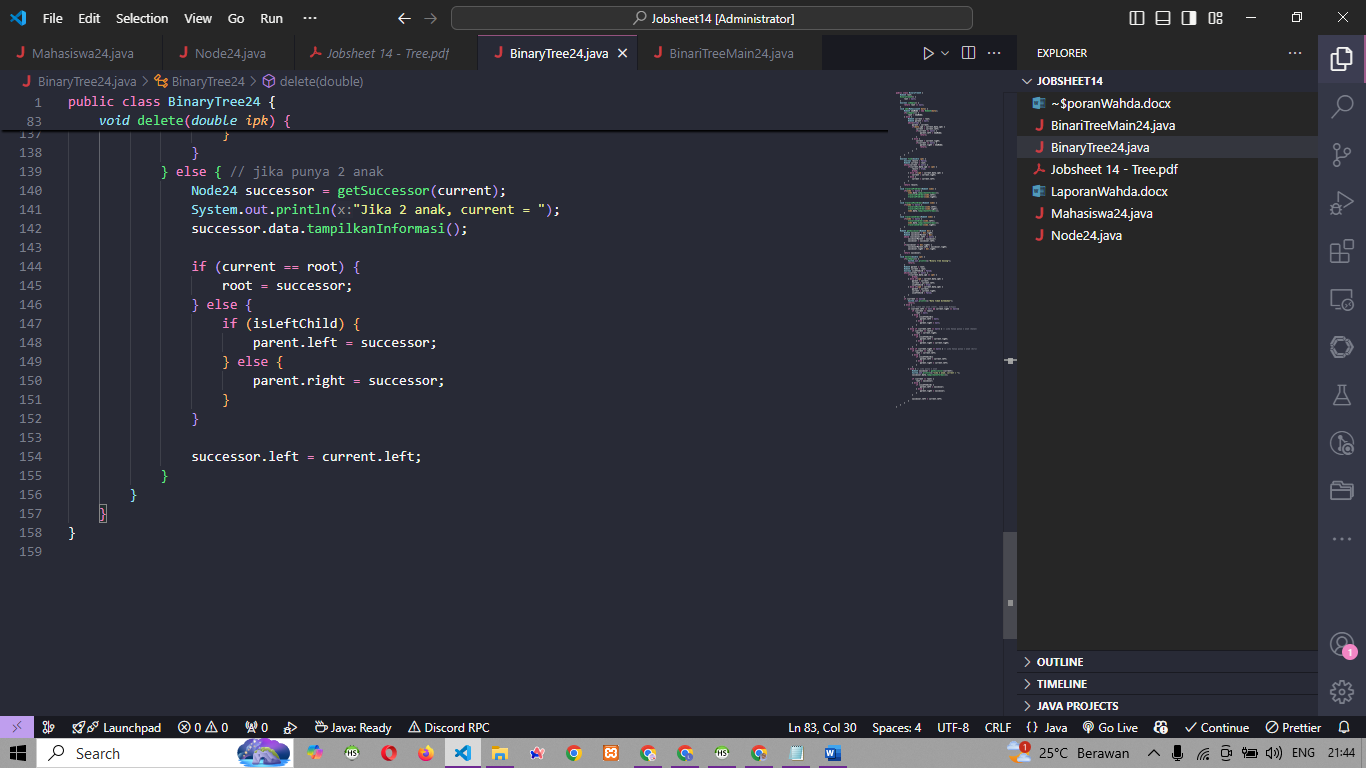
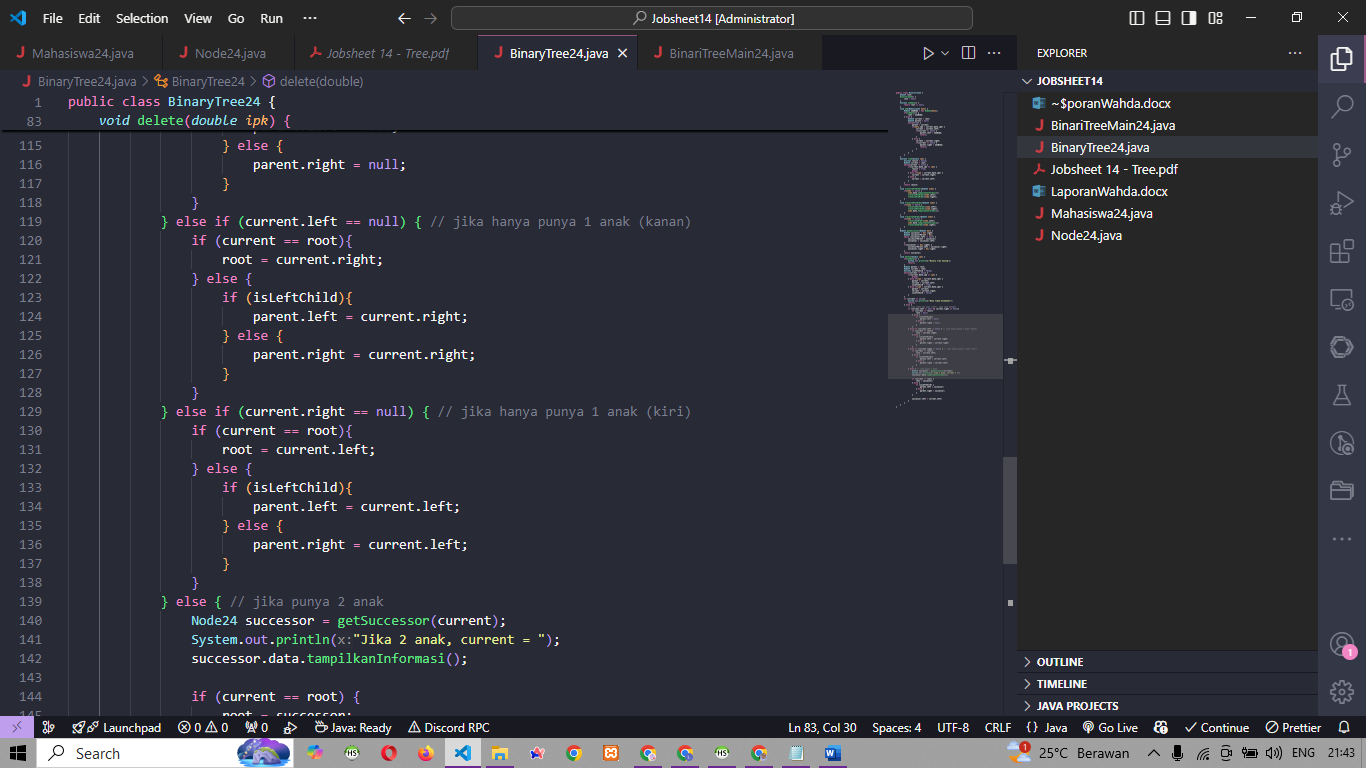
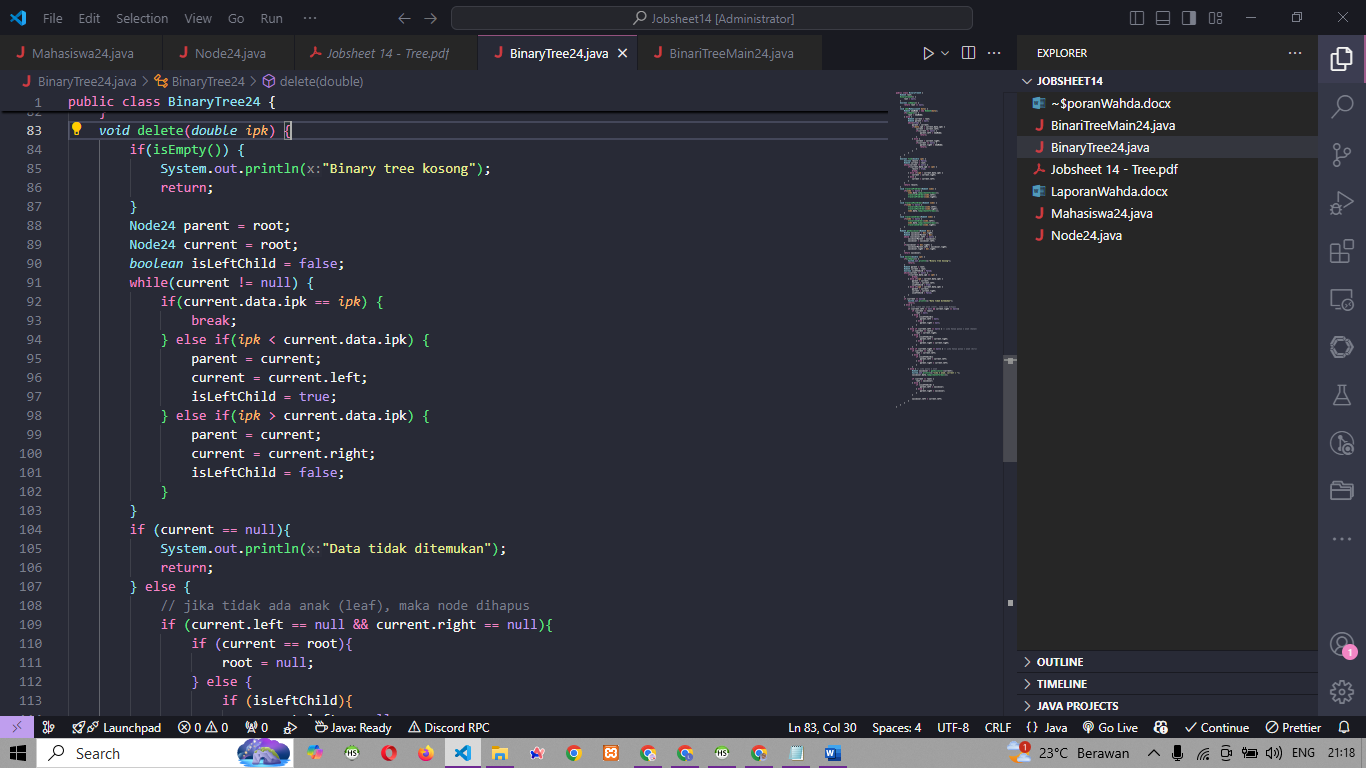
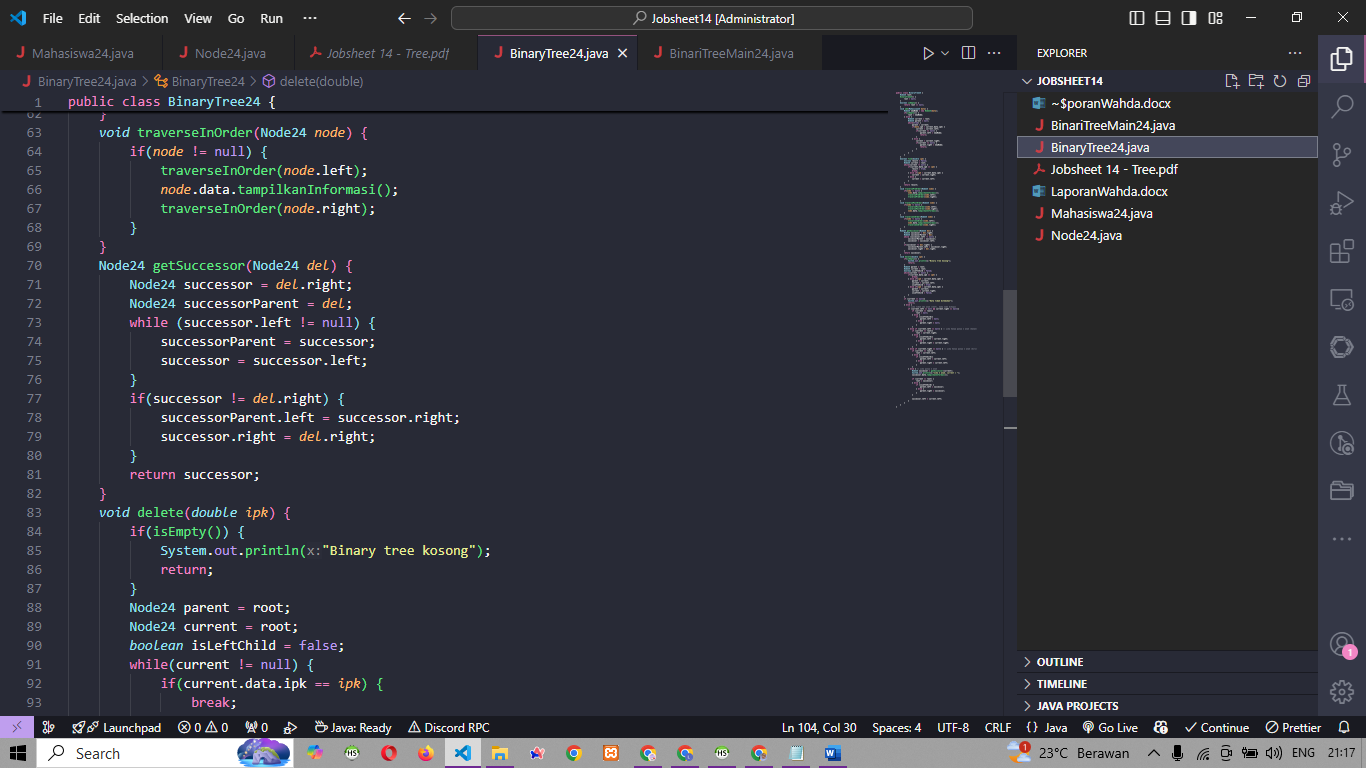
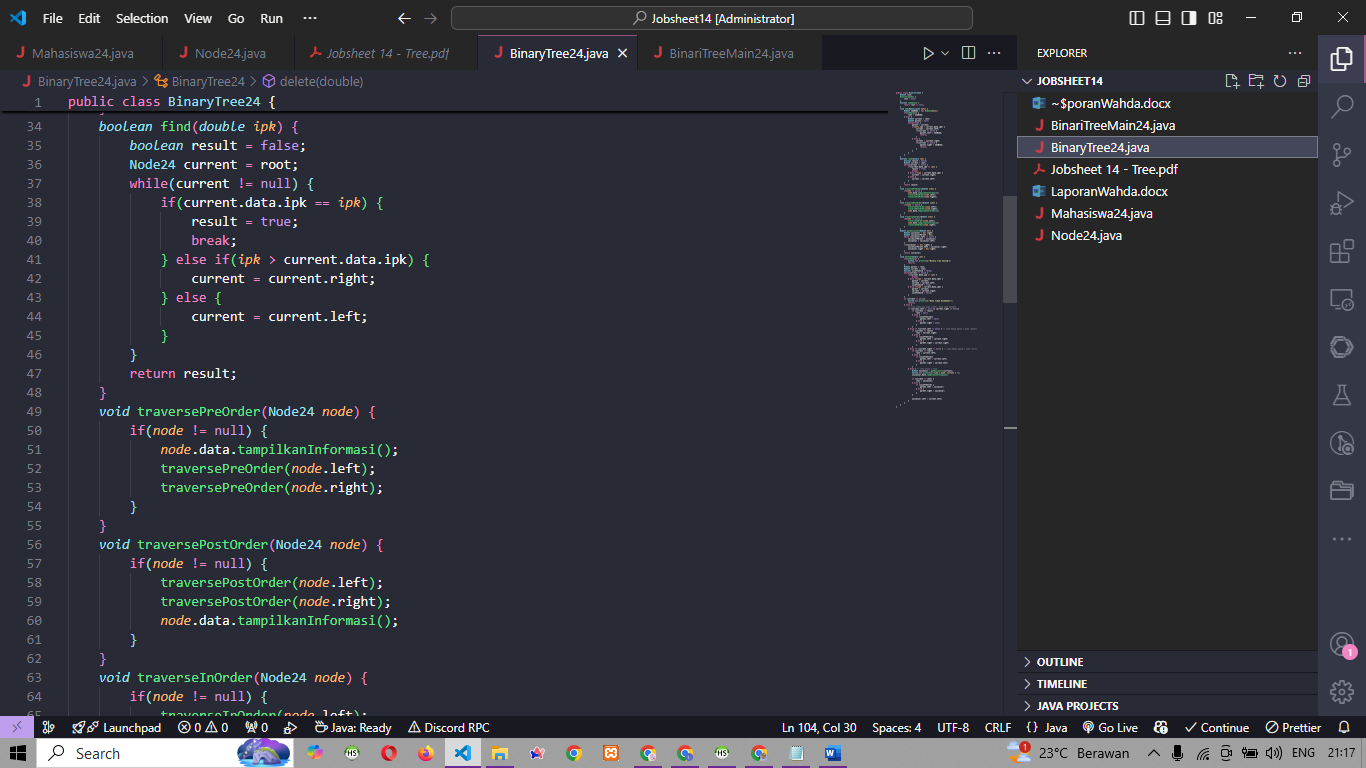
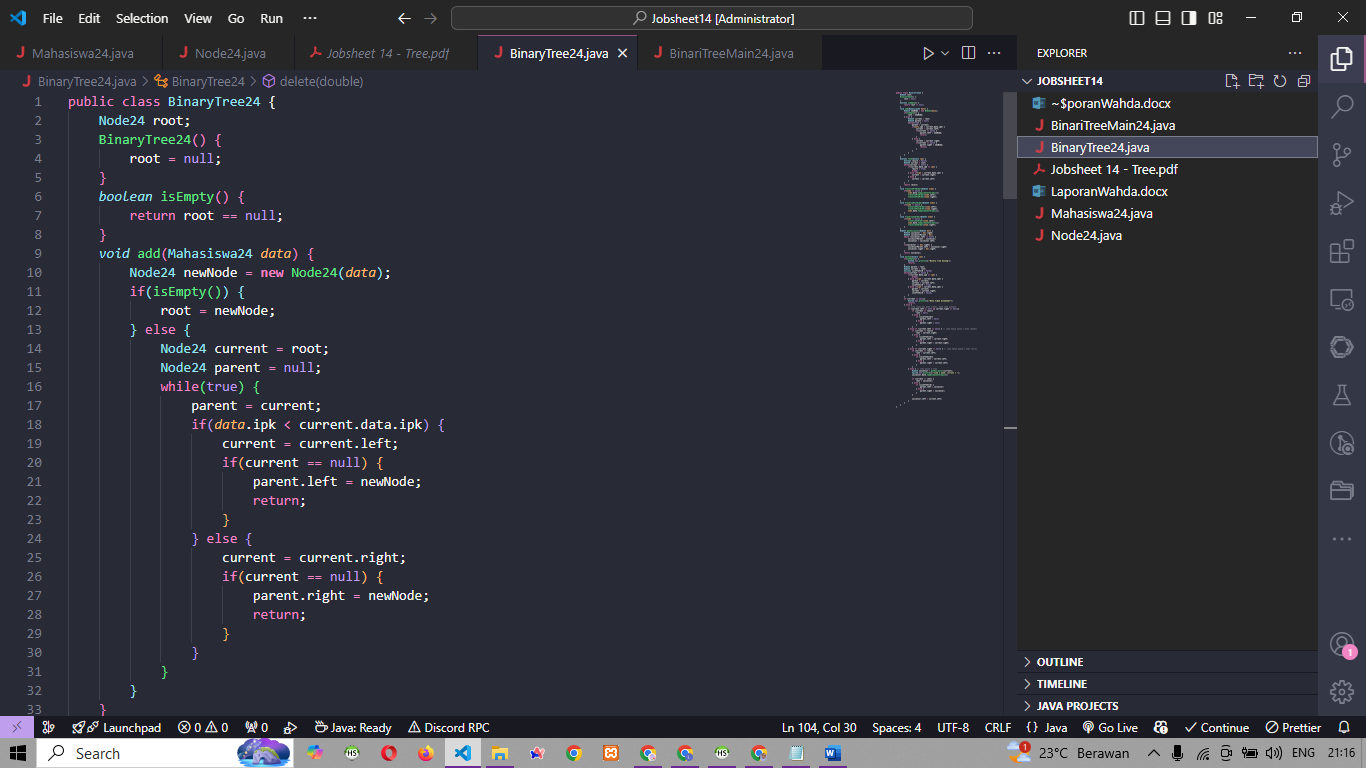
1. Buat file Mahasiswa24.java



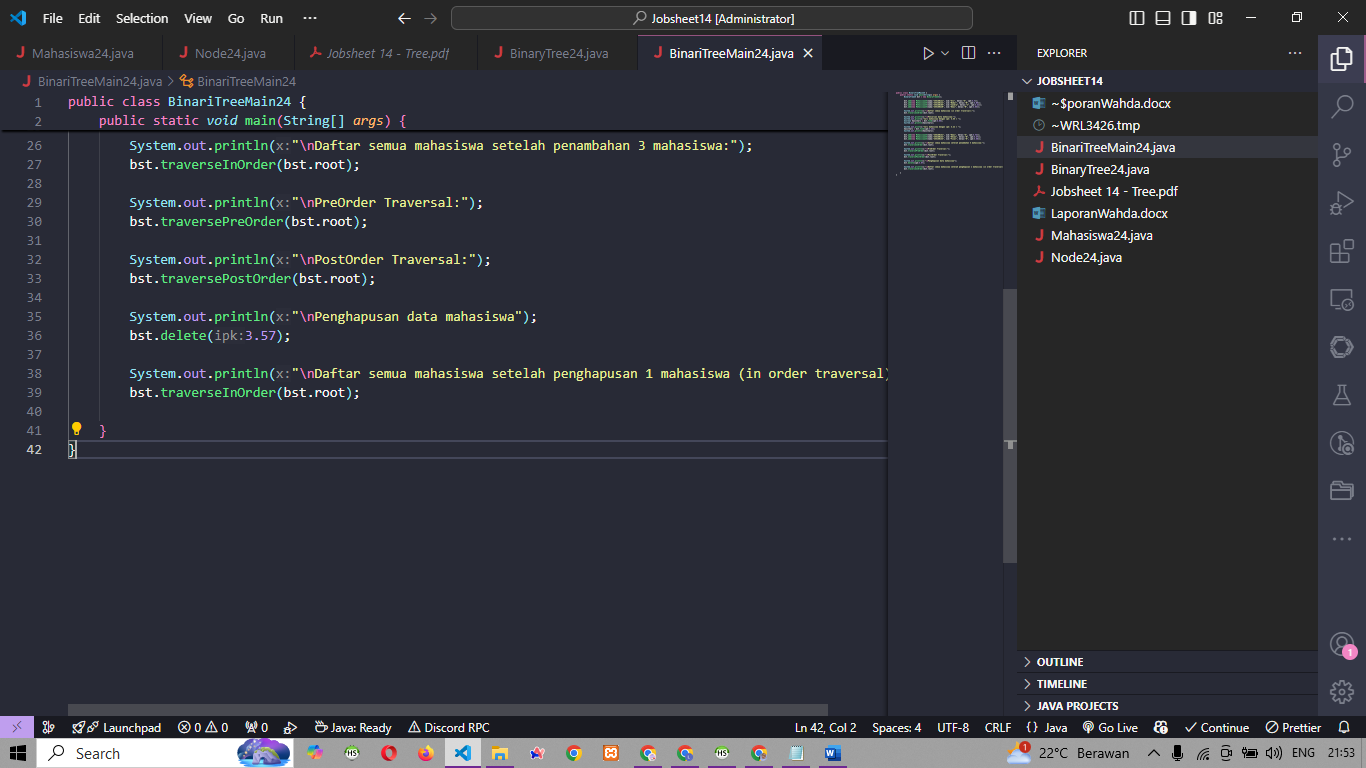
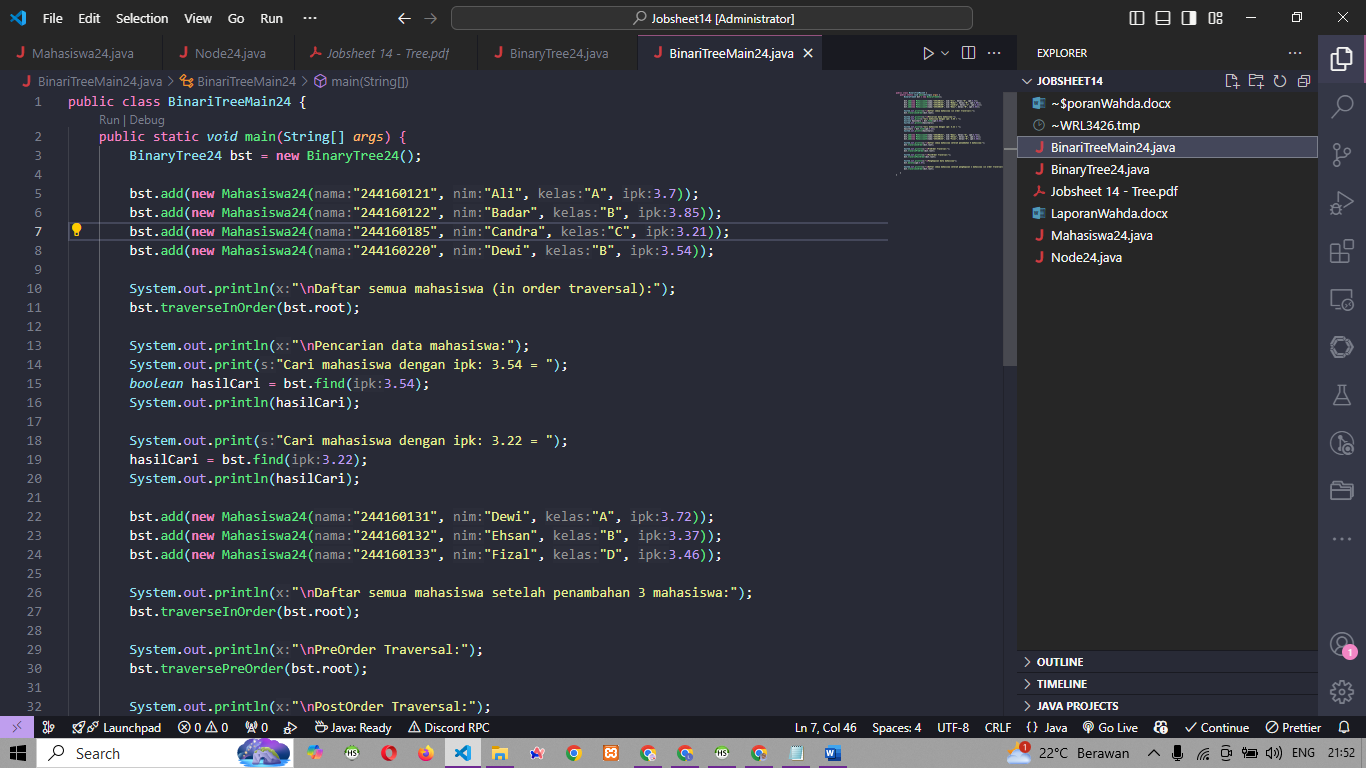
1. Buat file Node24.java



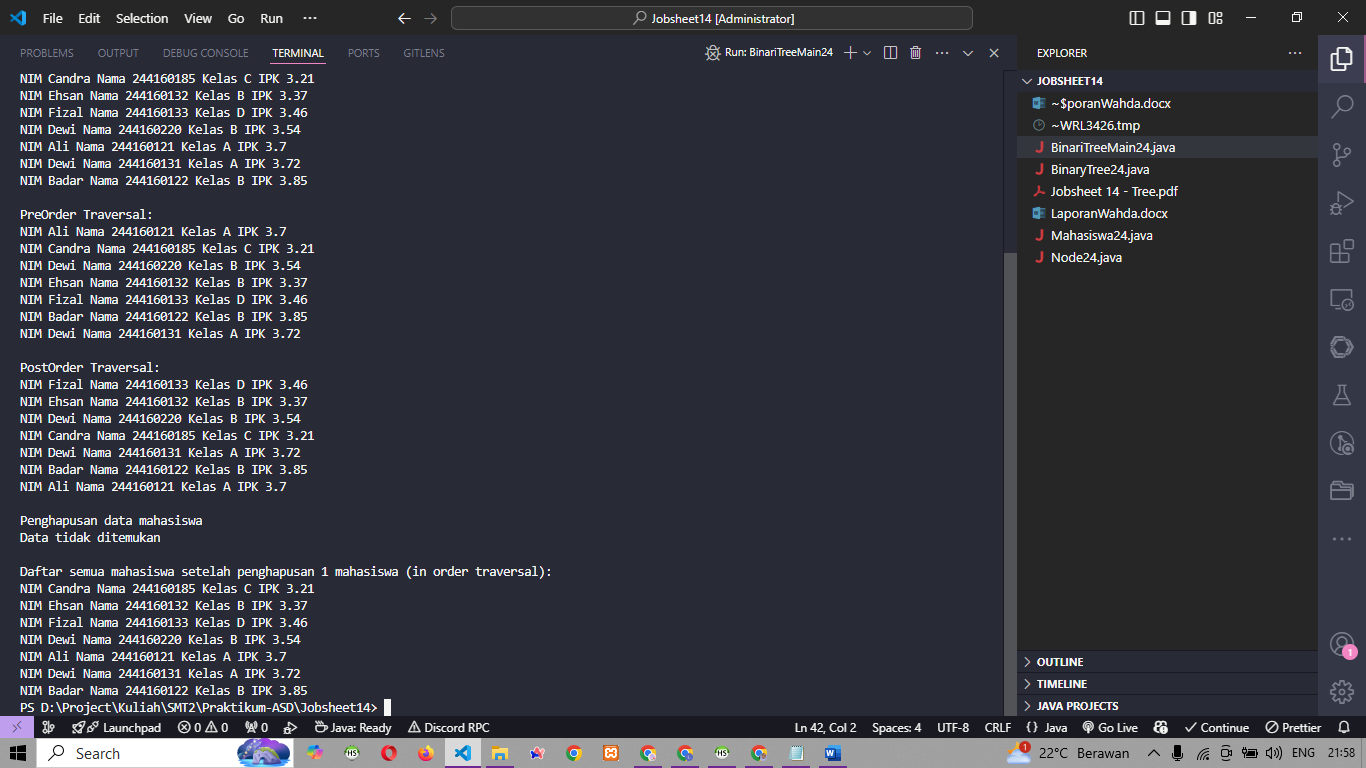
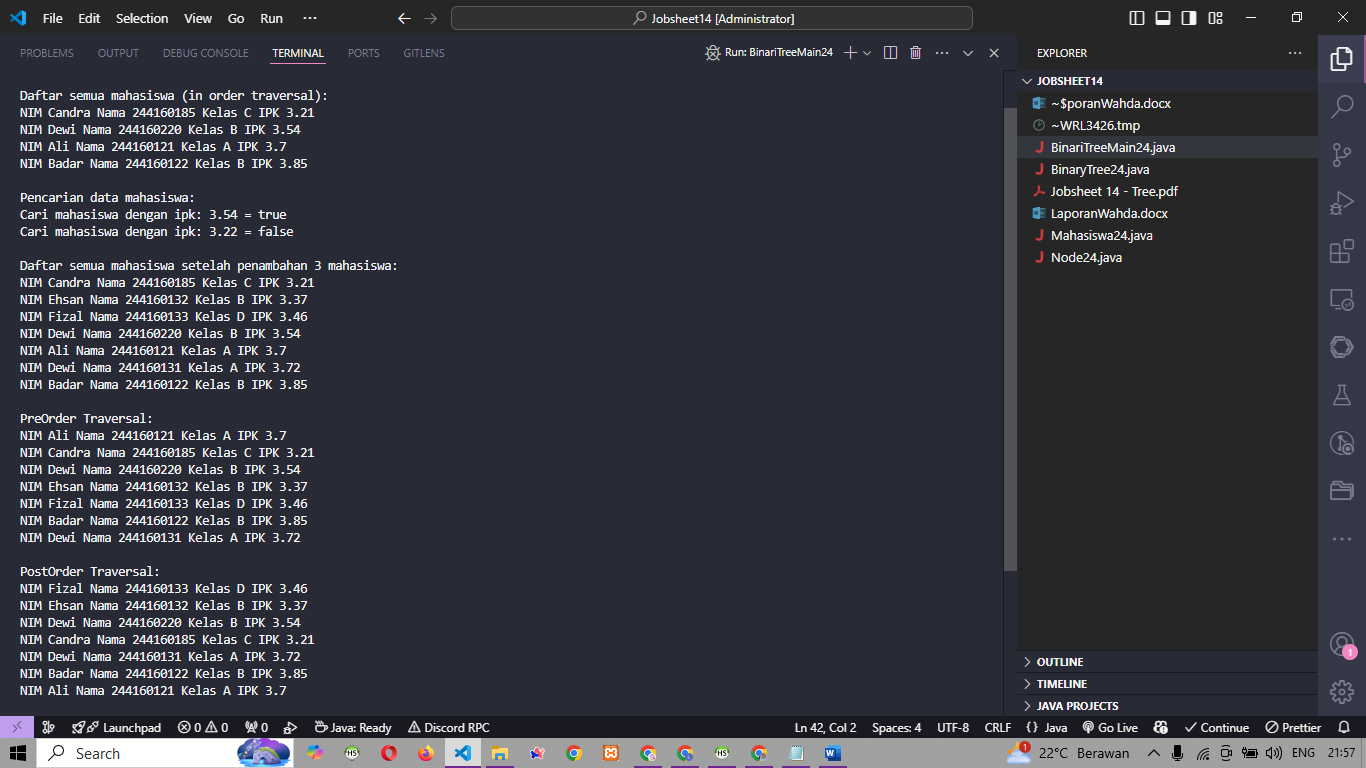
1. Buat file BinaryTree24.java



1. Buat file BinaryTreeMain24.java



1. Hasil Run



**Pertanyaan**

1. Karena dalam **Binary Search Tree (BST)**, data tersusun secara terurut:

* Node kiri selalu lebih kecil dari node induk.
* Node kanan selalu lebih besar dari node induk.

Dengan struktur ini, proses pencarian hanya perlu **membandingkan dan berpindah ke kiri atau kanan** secara logaritmik, sehingga kompleksitas waktu rata-rata adalah **O(log n)**.  
Sementara pada binary tree biasa, pencarian bisa memakan waktu **O(n)** karena tidak ada aturan urutan data, sehingga harus menelusuri seluruh node.

1. **left**: Menunjuk ke **anak kiri** dari node tersebut. Berisi data yang lebih kecil dari node induk.

**right**: Menunjuk ke **anak kanan** dari node tersebut. Berisi data yang lebih besar dari node induk.

1. A. root digunakan sebagai **titik awal** atau **node pertama** dari pohon biner. Semua operasi seperti add(), find(), delete(), dan traverse() dimulai dari node ini.

B. Saat pertama kali objek tree dibuat, nilai root adalah **null** karena pohon masih kosong dan belum ada data yang ditambahkan.

1. Jika tree kosong (root == null), maka:

* Node baru akan **langsung menjadi root**.
* Tidak perlu traversal karena tidak ada node lain.

1. Penjelasan:

* Jika mahasiswa.ipk lebih kecil dari current.ipk, maka pindah ke anak kiri (current = current.left).
* Jika current.left masih kosong (null), maka node baru (newNode) dimasukkan ke posisi tersebut (parent.left = newNode) dan proses berhenti (return).
* Kode ini menunjukkan **proses rekursif pencarian tempat kosong di anak kiri** untuk menyisipkan node baru pada Binary Search Tree.

1. **Langkah-langkah delete() untuk node dengan 2 anak:**

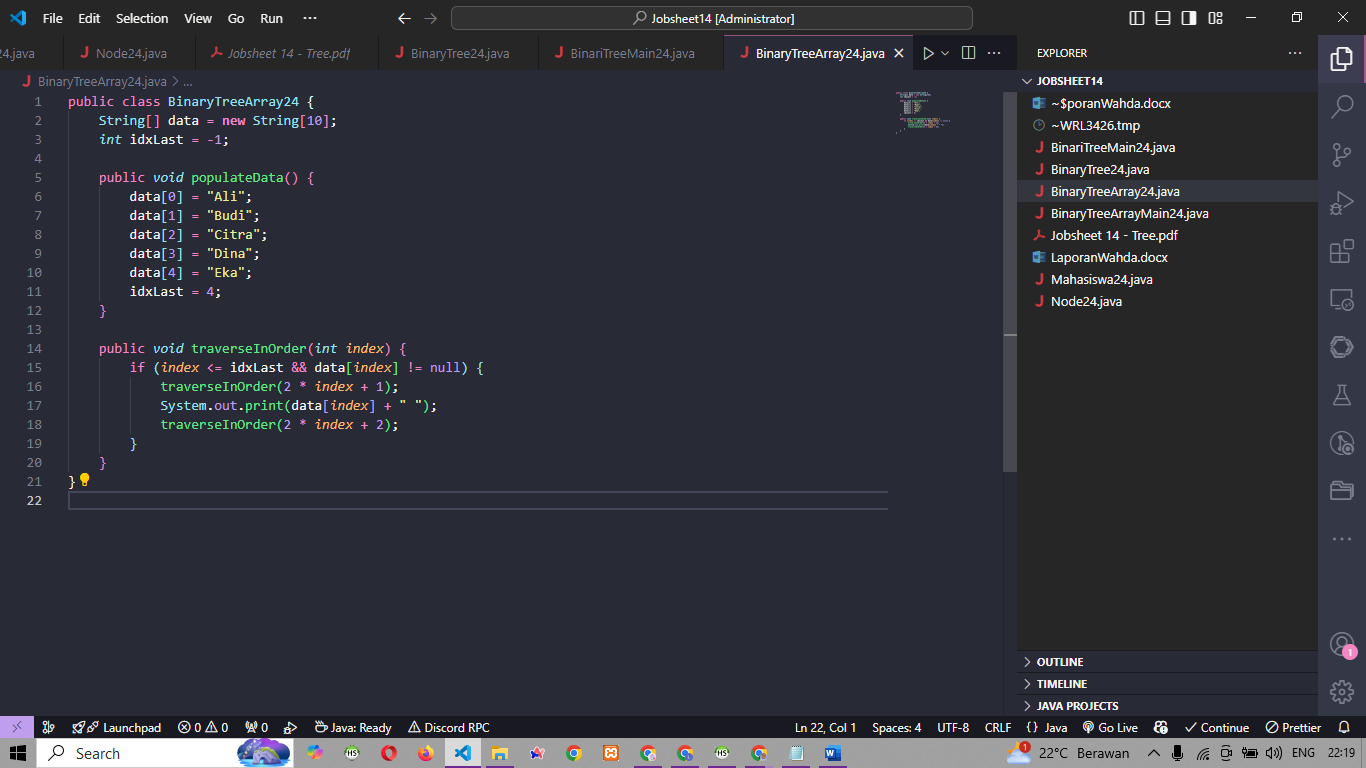
* Cari **successor** dari node yang akan dihapus, yaitu node **paling kecil dari subtree kanan**.
* Gantikan node yang ingin dihapus dengan successor.
* Pastikan successor.left diisi dengan current.left (anak kiri dari node yang dihapus).
* Jika successor bukan anak langsung dari node yang dihapus, maka pindahkan successor.right ke posisi successor sebelumnya.

**Peran getSuccessor()**:

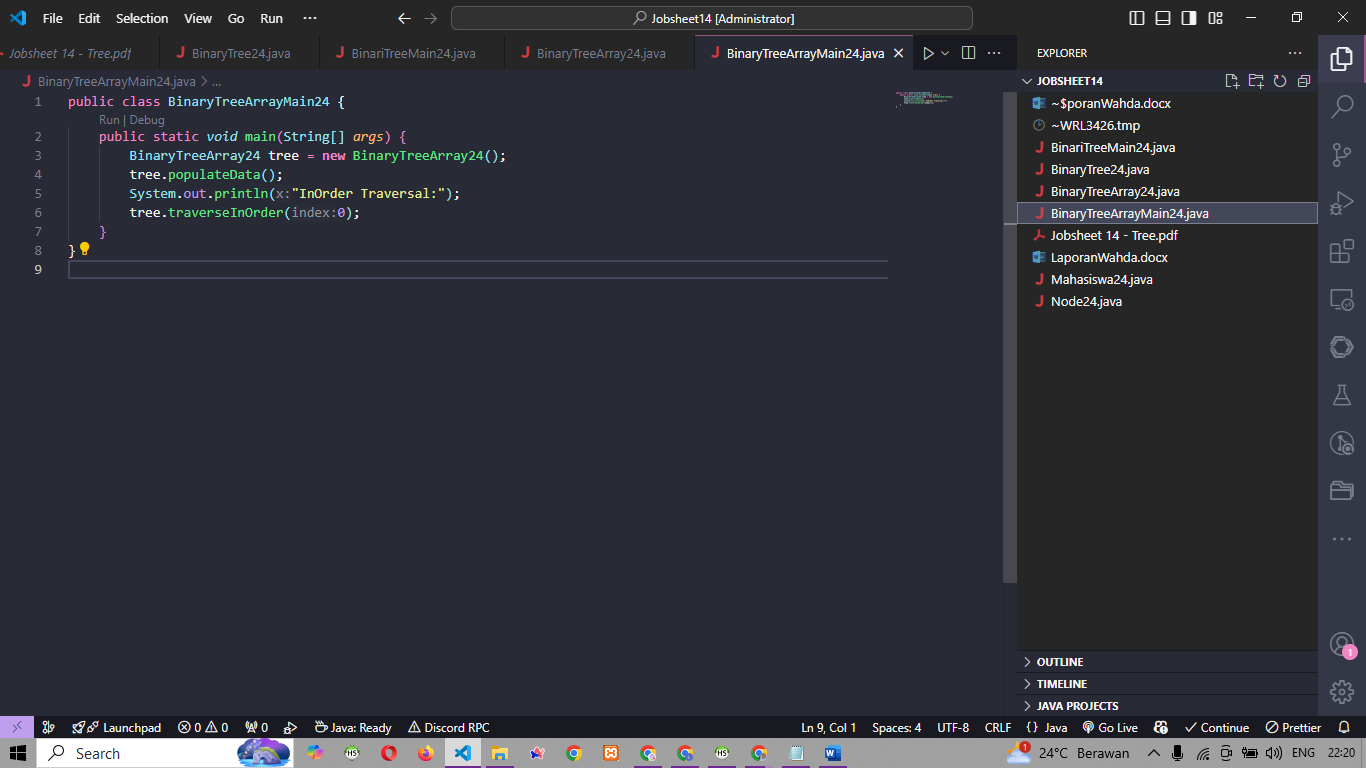
* Mengembalikan node yang paling layak menggantikan node yang dihapus agar **sifat BST tetap terjaga**.
* Menjamin bahwa struktur dan aturan BST tetap valid setelah penghapusan node yang punya 2 anak.

**Praktikum 2**

1. Buat file BinaryTreeArray24.java



1. Buat file BInaryTreeArrayMain24.java



1. Hasil Run

